機能拡張手続書

この文書では、フレームワークの機能を拡張するためのポイントとなる
仕組みや概念についてまとめる。

フレームワークの Ruby コードによる拡張

フレームワークには Ruby のコードで拡張できる2つの機能がある:
* ビジネスロジック
* 疑似項目

ビジネスロジック

ビジネスロジックインジェクションに利用されるライブラリは
    lib/business_logic
に配置する。例として
    test/mocks/test/audit_logic.rb
で DomainLogic を定義している。

ビジネスロジックインジェクションの実装はそれぞれ1つのクラスにし、
クラス BusinessLogic を継承する。
クラス名には対象となるモデルの名前 X を接頭辞に使って
   XLogic
にする(例: モデル Domain のビジネスロジックなので DomainLogic)。
各インスタンスメソッドで処理を定義する:
   :preprocess
    事前処理
   :postprocess
    事後処理
これらのメソッドは引数の1つで処理対象となるオブジェクトを受け取り、
そのオブジェクトを返すことが期待されている。

疑似項目

疑似項目に利用されるライブラリは
   lib/item_pseudo
 に配置する。例として
   lib/item_pseudo/sample.rb
 でクラス ItemPseudo::Sample を定義している。

疑似項目の実装はそれぞれ1つのクラスにし、クラス ItemPseudo の下の名前空間を利用する。
まずコンストラクタを定義する:
   :initialize
  このコンストラクタは1つの引数をとり、この引数には ItemPseudo オブジェクトが渡される。
そして疑似的に表示する内容を返すことが期待されているインスタンスメソッドを定義する:
   :to_data
  このメソッドは引数を1つとり、この引数には処理対象となるレコードが渡される。

Ruby on Rails

プラグイン

Ruby on Rails として配布されているライブラリ(ActionMailer, ActionPack, ActionWebservice,
ActiveRecord, ActiveSupport etc.)の機能を変更するといった拡張については、プラグインとして
追加する。
追加する際の規則は次の通り:
    * vendor/plugins ディレクトリにソースを配置する。
    * ソースファイル名は格納しているクラス名に対応させる。
      例: クラス FOOBarBaz をfoo_bar_baz.rb で定義する。

ライブラリ

Ruby on Rails に特化されていない拡張やプラグインに適さない拡張については、ライブラリとして追加する。
追加する際の規則は次の通り:
* lib ディレクトリにソースを配置する。
* ソースファイル名は格納しているクラス名と対応させる。
例: クラス FOOBarBaz をfoo_bar_baz.rb で定義する。

注意

基礎的な機能を変更する場合でも、(ActionController や ActiveRecord で行われているように)
新しいクラス/モジュールを追加して該当する実装を再定義する。
特に Rails のコードはそのままにしておく。
具体的な方法については
* lib/acts_as_attachable.rb
* lib/acts_as_relatable.rb
* lib/auto_image_tag.rb
* lib/quote_like.rb
* lib/user_system.rb
が参考になる。

スタイルシート

:.view_aspect
クラス (({view_aspect})) は詳細編集画面に付随するアスペクトを統一して扱うためのクラス。
現時点では
* 関連文書編集画面
* 添付ファイル編集画面
のスタイルを制御する。